```
#Sketch for data analysis side of VDJ sequences
#6 inputs for paired end reads and labels
#$1 is the diagnosis R1 fastq.gz file $2 is the diagnosis R2 fastq.gz file $3 is diagnosis label
#$4 is the BM/relapse R1 fastq.gz file $5 is the BM/relapse R2 fastq.gz file $6 is BM/relapse label

#Find VDJ matches
perl VDJmatch.pl --fastq1=$1  --fastq2=$2  --numbatch=1000 --outfile=$3_VDJmatches
perl VDJmatch.pl --fastq1=$4  --fastq2=$5  --numbatch=1000 --outfile=$6_VDJmatches
#Count VDJ matches

perl VDJcount.pl --vdjfile=$3_VDJmatches.txt --outfile=$3_VDJcount.txt
perl VDJcount.pl --vdjfile=$6_VDJmatches.txt --outfile=$6_VDJcount.txt
dVDJ=`tail -n 1 $3_VDJcount.txt | cut -f1`
rVDJ=`tail -n 1 $6_VDJcount.txt | cut -f1`

#Align both sets of matches to major VDJ in diagnosis sample
perl VDJalign.pl --vdjfile="$3_VDJmatches.txt" --outfile="$3_VDJ.aln" --vdj="$dVDJ"
perl VDJalign.pl --vdjfile="$6_VDJmatches.txt" --outfile="$6_VDJ.aln" --vdj="$dVDJ"

#Filter and sort both sets of alignments
perl VDJfilter.pl $3_VDJ.aln > $3_VDJ.aln.filtered;
perl VDJfilter.pl $6_VDJ.aln > $6_VDJ.aln.filtered;
perl VDJsort.pl $3_VDJ.aln.filtered > $3_VDJ.aln.filt.unq;
perl  VDJsort.pl $6_VDJ.aln.filtered > $6_VDJ.aln.filt.unq;

#Now create file of similar alignments
junc_one=$3_junctions_similar_to_$6.txt;
junc_two=$6_junctions_similar_to_$3.txt;

#now map similar alignments across the two samples
./VDJidentifySeqClosestToRef -refvdjfile $6_VDJ.aln.filt.unq -vdjfile $3_VDJ.aln.filt.unq -minclonefreq 10 | perl
~/PERL_SCRIPTS/sort_column_inv.pl 1 1 > $junc_one;
./VDJidentifySeqClosestToRef -refvdjfile $3_VDJ.aln.filt.unq -vdjfile $6_VDJ.aln.filt.unq -minclonefreq 10 | perl
~/PERL_SCRIPTS/sort_column_inv.pl 1 1 > $junc_two;
perl VDJselectSeqForPhylo.pl --vdjfile=$junc_one --vdj="$dVDJ" --label=$3 --labelo=$6 > $junc_one.fa;
perl VDJselectSeqForPhylo.pl --vdjfile=$junc_two --vdj="$dVDJ" --label=$6 --labelo=$3 --germline=0 >> $junc_one.fa;
perl -pi -e "s/>\n//" $junc_one.fa;
perl -pi -e "s/^\n//" $junc_one.fa;
perl TidyFASTA.pl --junctionfile=$junc_one.fa --outfile=$junc_one.fa.unq

#run multiple sequence alignment on similar alignments and generate .nwk file for tree building
clustalw-2.0.10/src/clustalw2 -infile=$junc_one.fa.unq;
# in R
R --no-save <<RSCRIPT
# R code
library(ape)
f <- "$junc_one.fa.aln";
m <- read.dna(f, format="clustal");
t <- nj(dist.dna(m));
t <- multi2di(root(t, "germline_1_"));
write.tree(t, file="$3$6_tree.nwk");
q()
RSCRIPT


##VDJmatch.pl script
#!/usr/bin/perl
use lib "$ENV{HOME}/PERL_MODULES";
use Sets;
use strict;

use Fasta;
use Sets;
use MyBlast;
use strict;

use Getopt::Long;
```

```
if (@ARGV == 0) {
  die "Args --fastq1=FILE --fastq2=FILE\n";
}

my $fastq2 = undef;
my $fastq1 = undef;
my $numbatch = 1000;
my $vdjdir = "$ENV{HOME}/PROGRAMS/VDJ";
my $ighvfile = "$vdjdir/DATA/IGHV-clean.fa";
my $vrefseq   = "$vdjdir/DATA/IGHV-clean.fa";
my $drefseq   = "$vdjdir/DATA/IGHD-clean.fa";
my $jrefseq   = "$vdjdir/DATA/IGHJ-clean.fa";
my $singleseq = undef;
my $verbose   = 0;
my $outfile   = undef;
my $numcpus   = 4;
my $debug     = 0;
my $stopafter = undef;

GetOptions("fastq1=s"    => \$fastq1,
                "vrefseq=s"   => \$vrefseq,
          "stopafter=s" => \$stopafter,
                "verbose=s"   => \$verbose,
                "debug=s"     => \$debug,
                "outfile=s"   => \$outfile,
                "numcpus=s"   => \$numcpus,
                "singleseq=s" => \$singleseq,
                "drefseq=s"   => \$drefseq,
                "jrefseq=s"   => \$jrefseq,
                "numbatch=s"  => \$numbatch,
          "fastq2=s"    => \$fastq2);



open IN1, "gunzip -c $fastq1 |" or die "Cannot open $fastq1\n";
open IN2, "gunzip -c $fastq2 |" or die "Cannot open $fastq2\n";

open OUT1, ">$outfile" or die "Cannot open $outfile\n";
open OUT2, ">$outfile.unmapped" or die "Cannot open $outfile.unmapped\n";

my $start = time;
my $totalcnt = 0;
my %CNT   = ();
my $exitnow = 0;
while (!$exitnow) {                              # master loop

  my $tmpfile = Sets::getTempFile("/tmp/tmpVDJ");
  open TMP, ">$tmpfile" or die "Cannot open $tmpfile\n";
  my $cnt = 0;
  my @a_reads = ();
  while (1) {

    my $l1_1 = <IN1>; if (!$l1_1) { $exitnow = 1; last; }
    my $l2_1 = <IN2>;
    my $l1_2 = <IN1>;
    my $l2_2 = <IN2>;
    my $l1_3 = <IN1>;
    my $l2_3 = <IN2>;
    my $l1_4 = <IN1>;
    my $l2_4 = <IN2>;

    chomp $l1_2;
    chomp $l2_2;
    chomp $l1_1;
    next if (defined($singleseq) && ($l1_1 ne $singleseq));
```

```
    $l2_2 = Sets::getComplement($l2_2);
    my $readseq = "$l1_2" . "NNNNNNNNNNNNNNNNNNNNNNNNNNNNN$l2_2";

    print TMP ">$l1_1\n$readseq\n";

    push @a_reads, [$l1_1, $readseq];

    $cnt ++;

    if ($cnt == $numbatch) {
      last;
    }
  }                                             # end this loop
  close TMP;

  my $h_ref_V = getTopBlastMatches($tmpfile, $vrefseq, 1, $numcpus, $verbose);
  my $h_ref_D = getTopBlastMatches($tmpfile, $drefseq, 1, $numcpus, $verbose);
  my $h_ref_J = getTopBlastMatches($tmpfile, $jrefseq, 1, $numcpus, $verbose);

  foreach my $r (@a_reads) {

    if (defined($h_ref_V->{$r->[0]}) && ($h_ref_V->{$r->[0]}->[4] > 5) && defined($h_ref_D->{$r->[0]}) && defined($h_ref_J->{$r->[0]})) {
      print OUT1 "$r->[0]\t$r->[1]\n";
      print OUT1 join("\t", @{$h_ref_V->{$r->[0]}}) . "\n";
      print OUT1 join("\t", @{$h_ref_D->{$r->[0]}}) . "\n";
      print OUT1 join("\t", @{$h_ref_J->{$r->[0]}}) . "\n";
      print OUT1 "\n";
      $CNT{"$h_ref_V->{$r->[0]}->[2] $h_ref_D->{$r->[0]}->[2] $h_ref_J->{$r->[0]}->[2]"} ++;
    } else {

      print OUT2 "$r->[0]\n";
      print OUT2 join("\t", @{$h_ref_V->{$r->[0]}}) . "\n" if defined($h_ref_V->{$r->[0]});
      print OUT2 join("\t", @{$h_ref_D->{$r->[0]}}) . "\n" if defined($h_ref_D->{$r->[0]});
      print OUT2 join("\t", @{$h_ref_J->{$r->[0]}}) . "\n" if defined($h_ref_J->{$r->[0]});
      print OUT2 "\n";

    }

  }



  $totalcnt += $cnt;

  my $time_taken = time - $start;

  if (($verbose == 1) && (($totalcnt % $numbatch) == 0)) {
    print STDERR "# processed $totalcnt in $time_taken seconds\n";
    foreach my $c (keys(%CNT)) {
      print "$c:$CNT{$c}    \n";
    }
    #print "        \r";
  }

  unlink $tmpfile if ($debug == 0);

   if (defined($stopafter) && ($totalcnt > $stopafter)) {
     last;
   }

} # while (1)

close OUT1;
close OUT2;

sub getTopBlastMatches {
```

```
 my ($tmpfile, $refseq, $evalue, $numcpus, $verbose) = @_;

 my $mb = MyBlast->new;
 $mb->setBlastProgram("blastn");

 $mb->setVerbose($verbose);
 $mb->setDatabaseDatabase($refseq);
 $mb->setNbProcessors($numcpus);
 $mb->setEvalueThreshold($evalue);
 $mb->setFilter(0);
 $mb->setGapOpening(2);
 $mb->setGapExtension(1);
 $mb->setq(-1);
 $mb->setWordLength(7);
 $mb->setQueryDatabase($tmpfile);

 if ($verbose ==1 ) {system("cat $tmpfile");}

 my $a_ref_reads = $mb->blastallMultiple_Unique(100); # blastall that returns top hit of each read, 100 HSP at most

 #open IN, $ARGV[0]; my @lines = <IN>; close IN;
 #my $txt = join("", @lines);

 #my $a_ref_reads = $mb->_analyzeMultiple_UniqueXML($txt, 100);

 #print "qname\tqlen\thitname\tevalue\tqfrom\tqto\tqframe\tdfrom\tdto\tdframe\n";


 my %res = ();  # results to return

 foreach my $r (@$a_ref_reads) {


   my $numhsps = @{$r->{HSPS}};
   my $hsp     = $r->{HSPS};

   # top match
   #my $i1 = Sets::min($hsp->[0]->{QFROM}, $hsp->[0]->{QTO});
   #my $i2 = Sets::max($hsp->[0]->{QFROM}, $hsp->[0]->{QTO});

   # size of matching region
   my $w  = abs($hsp->[0]->{QTO} - $hsp->[0]->{QFROM} + 1);
   my $fr = sprintf("%3.2f", 100*$w/$r->{QUERY}->{LENGTH});

   # identity
   my $dahsp = $mb->getHSPdata($hsp->[0]->{QSEQ}, $hsp->[0]->{DSEQ}); #  return [ $ug_qseq_len, $cnt_matches, $numins, $numdel ];
   my $id    = sprintf("%3.2f", 100*$dahsp->[1] / (length($hsp->[0]->{QSEQ}) - $dahsp->[2] - $dahsp->[3]));


   # else best match is ok
   # print hsp0
   my $txt = "$r->{QUERY}->{NAME}\t$r->{QUERY}->{LENGTH}\t$r->{HIT}->{NAME}";

   $res{$r->{QUERY}->{NAME}} = [ $r->{QUERY}->{NAME}, $r->{QUERY}->{LENGTH}, $r->{HIT}->{NAME}, $hsp->[0]->{EVALUE}, $fr, $id,
                                 $hsp->[0]->{QFROM}, $hsp->[0]->{QTO}, $hsp->[0]->{QFRAME},
                                 $hsp->[0]->{DFROM}, $hsp->[0]->{DTO}, $hsp->[0]->{DFRAME}  ];


 } # loop over reads, read from batch

 return \%res;
}

#VDJcount.pl script

#!/usr/bin/perl
use lib "$ENV{HOME}/PERL_MODULES";
```

```
use Sets;
use Fasta;
use bl2seq;

use strict;
 use Getopt::Long;

if (@ARGV == 0) {
  die "Args --vdjfile=FILE --f2=FILE\n";
}
my $vdjfile = undef;
my $vrefseq   = "$ENV{HOME}/DATA/IGHV-clean.fa";
my $drefseq   = "$ENV{HOME}/DATA/IGHD-clean.fa";
my $jrefseq   = "$ENV{HOME}/DATA/IGHJ-clean.fa";
my $vdj       = undef;
my $verbose   = 0;
my $showjunctions = 0;
my $outfile   = undef;
my $showname  = 0;
my $stopafter = undef;
my $singleseq = undef;

GetOptions("vdjfile=s"       => \$vdjfile,
                "verbose=s"       => \$verbose,
                "showname=s"      => \$showname,
                "singleseq=s"     => \$singleseq,
                "outfile=s"       => \$outfile,
                "stopafter=s"     => \$stopafter,
                "showjunctions=s" => \$showjunctions,
                "vdj=s"           => \$vdj);



# load ref sequences
my %V = ();
my %D = ();
my %J = ();
# load V, D, J ref seq
my $fa = Fasta->new;
$fa->setFile($vrefseq);
while (my $a_ref = $fa->nextSeq()) {
   my ($n, $s) = @$a_ref;
   $V{$n} = $s;
}
$fa->dispose;

$fa = Fasta->new;
$fa->setFile($drefseq);
while (my $a_ref = $fa->nextSeq()) {
   my ($n, $s) = @$a_ref;
   $D{$n} = $s;
}
$fa->dispose;

$fa = Fasta->new;
$fa->setFile($jrefseq);
while (my $a_ref = $fa->nextSeq()) {
   my ($n, $s) = @$a_ref;
   $J{$n} = $s;
}


my @aVDJ = ();

# count
open IN, $vdjfile or die "Cannot open $vdjfile\n";
my %CNT = ();
```

```
my $totalcount=0;
while (1) {
  my @a_l = ();
  my $l1 = <IN>; if (!$l1) { last; };
  my $l2 = <IN>;
  my $l3 = <IN>;
  my $l4 = <IN>;
  my $l5 = <IN>;

  chomp $l1;
  my @a_l1 = split /\t/, $l1, -1;

  chomp $l2;
  my @a_l2 = split /\t/, $l2, -1;

  chomp $l3;
  my @a_l3 = split /\t/, $l3, -1;

  chomp $l4;
  my @a_l4 = split /\t/, $l4, -1;

  $CNT{"$a_l2[2] $a_l3[2] $a_l4[2]"} ++;
  $totalcount++;
}
close IN;

open OUT, ">$outfile" or die "Cannot open $outfile\n";

my $a_ref_CNT = Sets::hash_order(\%CNT);
foreach my $r (@$a_ref_CNT) {
  my @a = split /\ /, $r;

  printf "%s\t%.0f\t%.2f%%\n",$r,$CNT{$r},$CNT{$r}/$totalcount*100;
  printf OUT "%s\t%.0f\t%.2f%%\n",$r,$CNT{$r},$CNT{$r}/$totalcount*100;
  #print "$V{$a[0]}\n";
}
close OUT;
print STDERR "# " . $a_ref_CNT->[ scalar(@$a_ref_CNT) - 1] . "\n";

## VDJAlign.pl script
#!/usr/bin/perl
use lib "$ENV{HOME}/PERL_MODULES";
use Sets;
use Fasta;
use bl2seq;

use strict;

use Getopt::Long;

if (@ARGV == 0) {
  die "Args --vdjfile=FILE --f2=FILE\n";
}
my $vdjfile = undef;
my $vrefseq   = "$ENV{HOME}/DATA/IGHV-clean.fa";
my $drefseq   = "$ENV{HOME}/DATA/IGHD-clean.fa";
my $jrefseq   = "$ENV{HOME}/DATA/IGHJ-clean.fa";
my $vdj       = undef;
my $verbose   = 0;
my $showjunctions = 0;
my $outfile   = undef;
my $showname  = 0;
my $stopafter = undef;
my $singleseq = undef;

GetOptions("vdjfile=s"       => \$vdjfile,
           "verbose=s"       => \$verbose,
```

```
                "showname=s"      => \$showname,
                "singleseq=s"     => \$singleseq,
                "outfile=s"       => \$outfile,
                "stopafter=s"     => \$stopafter,
                "showjunctions=s" => \$showjunctions,
                "vdj=s"           => \$vdj);



# load ref sequences
my %V = ();
my %D = ();
my %J = ();
# load V, D, J ref seq
my $fa = Fasta->new;
$fa->setFile($vrefseq);
while (my $a_ref = $fa->nextSeq()) {
   my ($n, $s) = @$a_ref;
   $V{$n} = $s;
}
$fa->dispose;

$fa = Fasta->new;
$fa->setFile($drefseq);
while (my $a_ref = $fa->nextSeq()) {
   my ($n, $s) = @$a_ref;
   $D{$n} = $s;
}
$fa->dispose;

$fa = Fasta->new;
$fa->setFile($jrefseq);
while (my $a_ref = $fa->nextSeq()) {
   my ($n, $s) = @$a_ref;
   $J{$n} = $s;
}


my @aVDJ = ();

if (!defined($vdj)) {
  # count
  open IN, $vdjfile or die "Cannot open $vdjfile\n";
  my %CNT = ();
  while (1) {
    my @a_l = ();
    my $l1 = <IN>; if (!$l1) { last; };
    my $l2 = <IN>;
    my $l3 = <IN>;
    my $l4 = <IN>;
    my $l5 = <IN>;

    chomp $l1;
    my @a_l1 = split /\t/, $l1, -1;

    chomp $l2;
    my @a_l2 = split /\t/, $l2, -1;

    chomp $l3;
    my @a_l3 = split /\t/, $l3, -1;

    chomp $l4;
    my @a_l4 = split /\t/, $l4, -1;

    $CNT{"$a_l2[2] $a_l3[2] $a_l4[2]"} ++;

  }
```

```
  close IN;

  my $a_ref_CNT = Sets::hash_order(\%CNT);
  foreach my $r (@$a_ref_CNT) {
    my @a = split /\ /, $r;

    #print "$r\t$CNT{$r}\n";
    #print "$V{$a[0]}\n";
  }

  print STDERR "# " . $a_ref_CNT->[ scalar(@$a_ref_CNT) - 1] . "\n";

  # create ref files
  @aVDJ = split /\ /, $a_ref_CNT->[ scalar(@$a_ref_CNT) - 1];



} else {


  @aVDJ = split /\ /, $vdj;

}


# V
my $tmprefseqV = Sets::getTempFile("/tmp/V");
open O, ">$tmprefseqV" or die "cannot open $tmprefseqV\n";
print O ">$aVDJ[0]\n$V{$aVDJ[0]}\n";
my $seqV = $V{$aVDJ[0]};
close O;
#system("cat $tmprefseqV");
my $lenV = length($V{$aVDJ[0]});

# D
my $tmprefseqD = Sets::getTempFile("/tmp/D");
open O, ">$tmprefseqD" or die "cannot open $tmprefseqD\n";
print O ">$aVDJ[1]\n$D{$aVDJ[1]}\n";
close O;
#system("cat $tmprefseqD");
my $lenD = length($D{$aVDJ[1]});

# J
my $tmprefseqJ = Sets::getTempFile("/tmp/J");
open O, ">$tmprefseqJ" or die "cannot open $tmprefseqJ\n";
print O ">$aVDJ[2]\n$J{$aVDJ[2]}\n";
close O;
#system("cat $tmprefseqJ");
my $lenJ = length($J{$aVDJ[2]});


open OUT, ">$outfile" or die "cannot open $outfile\n";

my $cnt_processed = 0;
open IN, $vdjfile or die "Cannot open $vdjfile\n";
while (1) {
  my @a_l = ();
  my $l1 = <IN>; if (!$l1) { last; };
  my $l2 = <IN>;
  my $l3 = <IN>;
  my $l4 = <IN>;
  my $l5 = <IN>;

  chomp $l1;
  my @a_l1 = split /\t/, $l1, -1;

  chomp $l2;
```

```
my @a_l2 = split /\t/, $l2, -1;

chomp $l3;
my @a_l3 = split /\t/, $l3, -1;

chomp $l4;
my @a_l4 = split /\t/, $l4, -1;

my $seq = $a_l1[1];
if ($a_l2[11] == -1) {
  $seq = Sets::getComplement($seq);
} elsif ($a_l2[11] != 1) {
  die "Prob\n";
}

# compare V to ref
my $thisv = $a_l2[2]; $thisv =~ s/\*.+$//;
if ($aVDJ[0] !~ /$thisv/) {
  next;
}
# compare D to ref
my $thisd = $a_l3[2]; $thisd =~ s/\*.+$//;
if ($aVDJ[1] !~ /$thisd/) {
  next;
}
# compare J to ref
my $thisj = $a_l4[2]; $thisj =~ s/\*.+$//;
if ($aVDJ[2] !~ /$thisj/) {
  next;
}




next if (defined($singleseq) && ($a_l1[0] ne $singleseq));

if ($showname == 1) {
  print OUT "$a_l1[0]\t";
}

#
# realign V region
#
my $tmpfile = Sets::getTempFile("/tmp/aVDJ");
open O, ">$tmpfile" or die "cannot open $tmpfile\n";
print O ">$a_l1[0]\n$seq\n";
close O;

my $bl = bl2seq->new;
$bl->setVerbose($verbose);
$bl->setD(0);
my $a_ref_m1 = $bl->bl2seq($tmpfile, $tmprefseqV);
my ($qseq, $dseq, $qstV, $qenV, $dstV) = @$a_ref_m1;

my @a_qseq = split //, $qseq;
my @a_dseq = split //, $dseq;

#print "$a_l2[2] $a_l3[2] $a_l4[2]\n";
#print "$qseq\n";
#print "$dseq\n";

my $idx = $dstV-1;
print OUT "-" x ($dstV-1);  # match starts at dstV in ref V
for (my $i=0; $i<@a_qseq; $i++) {
  if ($a_dseq[$i] ne '-') {
```

```
    if ($a_qseq[$i] eq 'n') {
            print OUT "-";
    } elsif ($a_qseq[$i] ne $a_dseq[$i]) {
            print OUT $a_qseq[$i];
    } else {
            print OUT ".";
    }
    $idx++;
  }
}
print OUT "-" x ($lenV - $idx);
# end V



#
# realign D region
#
print OUT " ";
my $bl = bl2seq->new;
$bl->setVerbose($verbose);
$bl->setD(0);
my $a_ref_m1 = $bl->bl2seq($tmpfile, $tmprefseqD);
my ($qseq, $dseq, $qstD, $qenD, $dstD) = @$a_ref_m1;


if ($showjunctions == 1) {
  # junction
  #print "$qenV\t$qstD\t$qenD\n";
  print OUT substr($seq, $qenV, $qstD-$qenV);
}

if ($qseq ne "") {
  my @a_qseq = split //, $qseq;
  my @a_dseq = split //, $dseq;

  #print "$a_l2[2] $a_l3[2] $a_l4[2]\n";
  #print "$qseq\n";
  #print "$dseq\n";

  my $idx = $dstD-1;
  print OUT "-" x ($dstD-1);
  for (my $i=0; $i<@a_qseq; $i++) {
    if ($a_dseq[$i] ne '-') {

            if ($a_qseq[$i] eq 'n') {
              print OUT "-";
            } elsif ($a_qseq[$i] ne $a_dseq[$i]) {
              print OUT $a_qseq[$i];
            } else {
              print OUT ".";
            }
            $idx++;
    }
  }
  print OUT "-" x ($lenD - $idx);
} else {
  print OUT "-" x $lenD;
}


#
# realign J region
#
print OUT " ";
my $bl = bl2seq->new;
```

```
 $bl->setVerbose($verbose);
 $bl->setD(0);
 my $a_ref_m1 = $bl->bl2seq($tmpfile, $tmprefseqJ);
 my ($qseq, $dseq, $qstJ, $qenJ, $dstJ) = @$a_ref_m1;

 if ($qseq ne "") {

  my @a_qseq = split //, $qseq;
  my @a_dseq = split //, $dseq;

  #print "$a_l2[2] $a_l3[2] $a_l4[2]\n";
  #print "$qseq\n";
  #print "$dseq\n";

  my $idx = $dstJ-1;
  print OUT "-" x ($dstJ-1);
  for (my $i=0; $i<@a_qseq; $i++) {
   if ($a_dseq[$i] ne '-') {

            if ($a_qseq[$i] eq 'n') {
              print OUT "-";
            } elsif ($a_qseq[$i] ne $a_dseq[$i]) {
              print OUT $a_qseq[$i];
            } else {
              print OUT ".";
            }
            $idx++;
   }
  }
  print OUT "-" x ($lenJ - $idx);
 } else {
  print OUT "-" x $lenJ;
 }

 print OUT "\n";

  #print join("\t", @$a_ref_m1) . "\n";


 #foreach my $r (@$a_ref_m1) {
 #  print join("\t", @$r) . "\n";
 #  last;
 #}
 #unlink $tmpfile;

 $cnt_processed++;

 if (($cnt_processed % 100) == 0) {
  print STDERR "# processed $cnt_processed            \r";
 }

 if (defined($stopafter) && ($cnt_processed >= $stopafter)) {
  last;
 }

 if (defined($singleseq) && ($a_l1[0] eq $singleseq)) {
  last;
 }


}
close IN;

##VDJfilter.pl script
#!/usr/bin/perl
use lib "$ENV{HOME}/PERL_MODULES";
use Sets;
```

```
use strict;

my @LIST = ();

open IN, $ARGV[0] or die "Cannot open $ARGV[0]\n";
my $num = 0;
while (my $l = <IN>) {
  chomp $l;

  $num++;
  #if (($num % 1000) == 0) {
  #  print STDERR "# $num reads processed         \r";
  #}
  next if (Sets::VDJsimilarity($l) < 0.8);

  print "$l\n";

}
close IN;
```

##VDJsort.pl script

```
#!/usr/bin/perl
use lib "$ENV{HOME}/PERL_MODULES";
use Sets;
use strict;
use Getopt::Long;


my @LIST = ();


open IN, $ARGV[0] or die "Cannot open $ARGV[0]\n";


my $num = 0;
while (my $l = <IN>) {
chomp $l;

  $num++;
  #if (($num % 1000) == 0) {
  #  print STDERR "# $num reads processed         \r";
  #}
  #next if (Sets::VDJsimilarity($l) < 0.8);

  my @a = split /\t/, $l, -1;


  $l =~ s/\-/\./g;
  push @LIST, $l;

  #print "$l\n";

}
close IN;


@LIST = sort { $a cmp $b } @LIST;


my @SEQS = ();

my $cnt = 1;
my $pl  = shift @LIST;
foreach my $l (@LIST) {
  if ($l ne $pl) {
    #print "$cnt\t$pl\n";
```

```
    push @SEQS, [$cnt, $pl];
    $pl = $l;
    $cnt = 1;
  } else {
    $cnt++;
  }
}


@SEQS = sort { $a->[0] <=> $b->[0] } @SEQS;

foreach my $s (@SEQS) {
        print "$s->[0]\t$s->[1]\n";
}


##VDJSelectSeqFor Phylo Script
#!/usr/bin/perl
use lib "$ENV{HOME}/PERL_MODULES";
use Sets;
use Fasta;
use bl2seq;

use strict;

use Getopt::Long;

if (@ARGV == 0) {
  die "Args --vdjfile=FILE --vdj=FILE\n";
}
my $vdjfile = undef;
my $vrefseq   = "$ENV{HOME}/DATA/IGHV-clean.fa";
my $drefseq   = "$ENV{HOME}/DATA/IGHD-clean.fa";
my $jrefseq   = "$ENV{HOME}/DATA/IGHJ-clean.fa";
my $vdj       = undef;
my $verbose   = 0;
my $showjunctions = 0;
my $outfile   = undef;
my $showname  = 0;
my $stopafter = undef;
my $singleseq = undef;
my $labelo    = "R";
my $label     = "D";
my $germline  = 1;

GetOptions("vdjfile=s"       => \$vdjfile,
           "verbose=s"       => \$verbose,
           "germline=s"      => \$germline,
           "label=s"         => \$label,
           "labelo=s"        => \$labelo,
           "showname=s"      => \$showname,
           "singleseq=s"     => \$singleseq,
           "outfile=s"       => \$outfile,
           "stopafter=s"     => \$stopafter,
           "showjunctions=s" => \$showjunctions,
           "vdj=s"           => \$vdj);



# load ref sequences
my %V = ();
my %D = ();
my %J = ();
# load V, D, J ref seq
my $fa = Fasta->new;
$fa->setFile($vrefseq);
while (my $a_ref = $fa->nextSeq()) {
```

```
    my ($n, $s) = @$a_ref;
    $V{$n} = $s;
}
$fa->dispose;

$fa = Fasta->new;
$fa->setFile($drefseq);
while (my $a_ref = $fa->nextSeq()) {
    my ($n, $s) = @$a_ref;
    $D{$n} = $s;
}
$fa->dispose;

$fa = Fasta->new;
$fa->setFile($jrefseq);
while (my $a_ref = $fa->nextSeq()) {
    my ($n, $s) = @$a_ref;
    $J{$n} = $s;
}


my @aVDJ = split /\ /, $vdj;

my $seqV = $V{$aVDJ[0]};
my $lenV = length($V{$aVDJ[0]});

my $seqD = $D{$aVDJ[1]};
my $lenD = length($D{$aVDJ[1]});

my $seqJ = $J{$aVDJ[2]};
my $lenJ = length($J{$aVDJ[2]});


my $cnt_processed = 0;
open IN, $vdjfile or die "Cannot open $vdjfile\n";

#print ">Relapse Major Clone\n$seqV$seqD$seqJ\n";
if ($germline == 1) {
  print ">germline(1)\n$seqV$seqD$seqJ\n";
}

my @ref = split //, "$seqV $seqD $seqJ";

# get rid of ref
my $l = <IN>;


# select N clones closest to ref
my $cnt = 0;
while ($cnt < 10) {
  my $l = <IN>;
  chomp $l;

  my @a = split /\t/, $l;

  # sequence
  my @b = split //, $a[4];

  my @newb = ();
  for (my $i=0; $i<@b; $i++) {
    if ($b[$i] =~ /[\.\-]/) {
      $b[$i] = $ref[$i];
      push @newb, uc($b[$i]);
    } elsif ($b[$i] ne " ") {
      push @newb, uc($b[$i]);
    }
  }
```

```
  $cnt ++;
  print ">$a[0]-$label" . "$labelo" . "like($a[3])\n" . join("", @newb) . "\n";
 }


# store all remaining clones
 my @CLONES = ();
 while (my $l = <IN>) {
  chomp $l;

  my @a = split /\t/, $l;

  # sequence
  my @b = split //, $a[4];

  my @newb = ();
  for (my $i=0; $i<@b; $i++) {
   if ($b[$i] =~ /[\.\-]/) {
    $b[$i] = $ref[$i];
    push @newb, uc($b[$i]);
   } elsif ($b[$i] ne " ") {
    push @newb, uc($b[$i]);
   }
  }

  $cnt ++;
  #print ">$a[0]-D (R-like) ($a[3])\n" . join("", @newb) . "\n";

  push @CLONES, [$a[3], "$a[0]-$label" . "major($a[3])",  join("", @newb) ];

 }

 @CLONES = sort { $b->[0] <=> $a->[0] } @CLONES;

 for (my $i=0; $i<10; $i++) {
  print ">$CLONES[$i]->[1]\n$CLONES[$i]->[2]\n";
  shift @CLONES;
 }

 srand(1234);
 my $n = @CLONES;
 for (my $i=0; $i<10; $i++) {
  my $m = rand($n)+0.5;
  $CLONES[$m]->[1] =~ s/major/random/;
  print ">$CLONES[$m]->[1]\n$CLONES[$m]->[2]\n";
 }
```